SHAPE

the SHarp Application Programming Environment

Preliminary Specifications

Gitte Christensen, Morten Kromberg
Helsingør, 1987-8-6

Copyright ©mcmlxxxvii I.P.Sharp ApS

TABLE OF CONTENTS

0. Introduction 1

1. Design Goals 2

2. Definition of Terms 3
- 2.1. Sites 3
- 2.2. Site Defaults 4
- 2.3. Systems 4
- 2.4. System Defaults 5
- 2.5. System Documentation 5
- 2.6. Phases 5
- 2.7. Variants 5
- 2.8. End Environments 5
- 2.9. Files 6
- 2.10. Data Types 7
- 2.11. Data Transformations 8
- 2.12. Data Validation 9
- 2.13. Profiles 9
- 2.14. Macros 9
- 2.15. Menus 10
- 2.16. Commands 11
- 2.17. PF-Keys 11
- 2.18. Screens 12
- 2.19. Screen States 13
- 2.20. Fields 13
- 2.21. Field Groups 14
- 2.22. Field Tables 14
- 2.23. Special Fields 14
- 2.24. Graphs and Images 14
- 2.25. Helps 14

3. Welcome to SHAPE 15

4. Systems, Phases, Variants and Environments 15
- 4.1. Systems 16
- 4.2. Variants 18
- 4.3. Environments 19
- 4.4. Access Control 20

5. Data Types, Transformations, Validation 22
- 5.1. Data Types 22
- 5.2. Transformations 24
- 5.3. Validations 24

6. Files 25
- 6.1. File Handling 25
- 6.2. Interfacing Files to Screens 26
- 6.3. File Design 27

6.4. APL File Utilities 29
7. Menus 30
7.1. How does SHAPE use Menus? 30
7.2. Shortcuts 30
7.3. Defining Menus 31
7.4. Layout of Menu Screens 32
8. Commands 33
8.1. Command Arguments 33
8.2. Commands and Screens 34
8.3. The Scope of Commands 34
9. APL Code 35
9.1. Modules 35
9.2. Reference to Screen Data 35
10. Screen Design 35
11. Helps and Tutorials 35
12. Graphics 36
13. Running the Application 36
13.1. Debugging 37
13.2. Dummy Files and Conditional File Updates 37
13.3. Error Logging, Restartability 37
14. Generating Documentation 38
15. Utility Functions 38
15.1. Print Spooling System 38
15.2. Batch Task Control System 38
15.3. Resource Consumption Monitor 39
16. System Administrator Duties 39
17. Macros 40
18. Ideas 40
18.1. Integration with VIEWPOINT 41
18.2. Conversion tools 41
18.3. Importing APE(2) applications 41
18.4. Producing BMS macros 41
18.5. Text Editor 41
18.6. PC Support 41
19. Desired extensions to LOGOS 42

Preface

Dear Reader,

Here are the Preliminary Specifications for SHAPE - The Sharp Application Programming Environment.

In writing these specifications, we have tried to avoid concentrating on implementation details, except where these have direct influence on underlying concepts. We have attempted to paint a complete picture of the functionality we would like to give SHAPE.

This approach should ensure that we can receive as much input from you as possible during the creative part of designing the product, and that our preliminary "brainstorming" sessions do not get bogged down in technical details. It should also avoid our rejecting otherwise good ideas because they do not fit in well with our implementation philosophy.

When we have reached agreement on the overall functionality of SHAPE, we hope that we can draw upon those among you who are experts in the various fields which SHAPE will try and bring together.

Happy reading!

Gitte Christensen and Morten Kromberg
Helsingør – August 6th, 1987

0. Introduction

The Sharp Application Programming Environment (SHAPE) is an interactive, menu-driven application generation tool. The project stems from a contract with a Danish savings bank, BIKUBEN, who demanded a system with (at least) the functionality of IBM's APE2, if they were to chose SAPL over APL2. Having heard several customers ask for this sort of system, both in and out of Denmark, we have proposed a joint venture with IPSA to develop a more general solution that that required by BIKUBEN.

SHAPE is basically a method of imposing a set of standards upon an APL programming department by providing a set of tools which is so attractive that the programmers have more to gain in ease of use than they have to lose in freedom. This has been attempted before - many times - so why do we have the arrogance to believe that we shall succeed where all others have failed? We believe that we will succeed because de-facto standards for both programming development methods and the functionality of fullscreen front-end are now emerging within IPSA in the form of LOGOS and VIEWPOINT.

SHAPE will be a sort of Active LOGOS User's Guide - it helps programmers use LOGOS in the "proper" way, which means the creation of systems which have VIEWPOINT's user-interface. making use of LOGOS paging files to implement systems which can have multiple TEST phases (in a controlled manner), and support multiple languages.

SHAPE will have all the tools required for implementing APL systems in a "Production Environment". This means that the release of a new phase of an application is controlled by a System Administrator, who has access to check that the required documentation for the system is present, and so on. In addition, ALL uses of files (if desired) can be controlled by SHAPE, using a table of file names to be used for each phase. Thus, SHAPE makes sure that TEST programmes only access TEST files.

In addition to being a tool for "professional" APL programmers, the Screen Painter and File Interface tools in SHAPE will allow professional people with little or no knowledge of Data Processing to implement simple systems in APL. With only an introductory course to APL they will be able to express their problem, without having to learn what "shared variable control" is. From our first customer's point of view, this is in fact the MAIN reason for requiring SHAPE, but we decided to mention the professional APL'ers first, considering the expected audience of this document.

This Document is the first draft of the specifications for SHAPE. The first release of SHAPE at this moment only 6 months away, and we can obviously not include everything described here in that release. We were planning to include completion dates for a phased implementation of SHAPE in this document, but it appears that the document has fallen prey to "phased implementation" as well.

Of course, SHAPE will be implemented using itself. Hopefully this means that the development team will do a good job - or they will themselves be subjected to any "design weaknesses" at an early stage.

1. Design Goals

The main design goal of SHAPE can be quite simply stated: to be everything to everyone developing fullscreen applications in SAPL. Some guidelines which we have tried to follow (ordered according to 8?8) are:

- Data Validation. In our experience, the most annoying parts of fullscreen application development without some sort of tools are **Designing the Screens**, and **Validating the Data**. This fact has been recognised by many developers of utilities, but for some reason most people seem to concentrate on Designing the Screens. Data Validation is perhaps even more annoying - writing 50-line functions consisting of branches to labels and the issuing of various error- messages depending on the results of various combinations of QUAD-VI's and QUAD-FI's, and a few ∧.= has got to be one of the most depressing tasks for an APL'er (one might even be tempted to say that APL is not a good language for expressing this sort of thing, if it were not for the fact that APL is good at everything).

- Ease of use for the naive as well as the sophisticated user, and for users with varying taste. This means that sophisticated features (for example multiple phases or languages) must be HIDDEN from the user who does not require them. There should be multiple paths to performing the same function. For example, attributes of fields on a screen should be editable by pointing at the field and "ZOOM"ing on it, by issuing a command, or via one or more attribute screens, which allow the user to see and edit attributes for many fields at once.

- Completeness. The amount of APL Code which needs to be written by the user should be reduced to an absolute minimum. For example, changing the type of a a field on a screen (from input to output) during the execution of an application should be done through an awareness in SHAPE of "Multiple Screen States", and not by calling a utility function at run-time.

- Producing applications with the a User Interface with the same features as VIEWPOINT (at least).

- Efficiency: the run-time portion of SHAPE (the screen driver and file interface utilities in particular) must be highly efficient. For example, pre-compilation of code should be used rather than 'executes' and run-time table lookups, wherever possible.

- Support for test and production phases, and multiple languages (Variants).

- Design of Dynamic Screens, which "automaticaly" adjust to fit Data and/or Physical Screen sizes.

- No requirement for an APL keyboard AT ANY TIME. This puts some constraints on Screen Design, and implies an APL keyword scheme of some sort for writing APL code on non-APL terminals.

2. Definition of terms

Possibly, this chapter should be an appendix, rather than appearing here at the beginning of the Document. You may wish to skip these "definitions of terms" and move straight to the more verbose explanations starting with chapter three. Not everything made its way into both parts of the specifications, so to see all our ideas (and to gain some insite into how we changed our minds as we went along) you should read everything.

This Chapter attempts to define the following terms:

Sites, Site Defaults, Systems,
System Defaults, System Documentation,
Phases, Variants, End Environments, Files,
Data Types, Data Transformations, Data Validations,
Profiles, Macros, Menus, Commands, PF-keys
Screens, Screen States,
Fields, Field Groups, Field Tables, Special Fields
Graphs and Images, Helps.

2.1. Sites

Installations of SHAPE on an APL system. Currently, SHAPE is only available to run under SAPL/370 on IBM and IBM-compatible mainframes.

2.2. Site Defaults

Site Defaults are a set of choices made at a site which control the way in which programmers at a site make use of some of the options in SHAPE. Some Site Defaults can be made COMPULSORY, i.e. the programmer may not deviate from the Default. Current Site Defaults are:

- Default PF-key settings (mappings of PF-keys to COMMANDS), for example PF3 for BACK, PF7 to UP and PF8 to DOWN.
- Default Colours and Highlights for:
  Input Fields, Output Fields, Fixed Text, Command Lines, Message Lines, Highlighting of Errors, Inserted and Deleted Lines in Scrolling Fields.
- Three Default Screens: One which defines the layout of Menus at the site, one which is used as a base for the definition of all new screens at a site to encourage similar screen layouts across applications, and a default layout for Help screens.
- A PROFILE, which is the default for all users added to the system.
- A site may define its own MACROS, COMMANDS, DATA TYPES, TRANSFORMATIONS and VALIDATIONS so that these are automaticaly available to all applications. APL-coded utility functions and variables may likewise be collected in a central utility library.
- At multilingual sites: Variants of certain character tables, such as the words 'YES' and 'NO', names of months and so on, as required for DATA VALIDATION. It may also be necessary to define translate tables for DATA TRANSFORMATIONS, for example when translating all input text to uppercase, National Language characters must be included in the tables. SHAPE will include tables for as many languages as possible (i.e. ENGLISH, DANSK).

  System Defaults are maintained by the System Administrator.

2.3. Systems

A SYSTEM is an Application being developed within SHAPE. Each SYSTEM has the following attributes:

- Name.
- Short Description.
- System Documentation.
- One or more Phases (for example TEST and PROD).
- One or more Variants (for example ENGLISH and DANSK).
- One or more End Environments.
- A Developer Access table: who may read or modify which combinations of PHASES and VARIANTS.
- A User Access table: which user-numbers (or ALIAS'es?) have access to which End Environments? This control can be given to a central facility maintained by the systems administrator, or even to an 'external' Access Control Facility such as RACF.

2.4. System Defaults

Each System may have its own Defaults, which override the Site Defaults, where this is permitted.

2.5. System Documentation

In this release, "Documentation" consists of character vectors which the programmer may edit to supplement the Short Descriptions of Systems (as VIEWPOINT has an extended File Description). In the future, we may want to add a more sophisticated Document Library System, with a nice Text Editor and so on.

2.6. Phases

A System can have one or more PHASES, representing various stages of development. Many (perhaps most) systems at a site will not be critical enough to warrant multi-phase development, and for these systems SHAPE will hide the fact that there could be multiple phases, so as not to frighten people unnecessarily.

A Typical multi-phase System will have a TEST phase and a PROD phase. The PROD phase will represent the current production system, and new development will be done on the TEST phase. More phases can be added if there are several projects under way to enhance various parts of the system, if it is desirable to test the different changes seperately (or in selected combinations).

In most cases, two End Environments will be generated: one from the PROD phase and one with objects from the TEST phase overriding any identically named objects in the PROD phase. When making changes to the system, objects will be picked up from the PROD phase if no object by a given name exists in TEST, but the modified object will be replaced in TEST.

For systems defined as "Production Systems", modification of the PROD phase will be prohibited by SHAPE. Only the System Adminstrator can authorize the copying of TEST objects into a PROD phase.

2.7. Variants

A System may have one or more VARIANTS, representing versions of the system which differ only in Text Constants. Variants are normally used to create multi-lingual systems, but one could also use the concept of variants to support different terminal types, for example Union versus old-style APL versions of a system.

2.8. End Environments

End Environments are in fact LOGOS paging environments, and a SHELL which fires up the selected Variant of the End Environment, displays the main menu of the application, and continues to run the entire application.

End Environments are generated by overlaying PHASES, simply by selecting multiple working directories withing LOGOS during the generation of the paging files.

End environments have the following attributes:

- Name.
- Short Description.
- List of PHASES to be overlaid, in order of precedence.
- A switch defining whether or not the Environment is a Production Environment.

Production Environments are a special class, where the release of new code into production is strictly controlled by the system administrator (or other authorised individuals).

2.9. Files

Files are various forms of Data Storage which can be accessed by applications maintained in SHAPE. SHAPE recognises a number of file types:

- DB2 and VIEWPOINT files accessed using SSQL and VPUTIL.
- VSAM files accessed through VSDI.
- OS/370 datasets accessed via TSIO.
- 'FLAT' APL files.
- FREE-FORM APL component files.

SHAPE contains utility functions for accessing all of the above. The FILE definition in SHAPE attaches a symbolic name to each file, which is used to refer to the files within an application. Thus, applications in SHAPE do not contain any file names in their code.

Files have the following attributes:

- Name.
- Short Description.
- File Documentation (as System Docn., just a character vector for now).
- Type (DB2/VUPT/APLFLAT/APLFREE etc).
- For all files except APLFREE, field definitions.
- An 'external' file name for PROD environments.
- An optional external file name for TEST environments.

2.10. Data Type

Data Types describe the data which is manipulated by applications maintained in SHAPE. Within SHAPE, a Data Type is defined as follows:

- A Name.
- Short description, and documentation.
- A parameter list.
- A "Base Data Type"
- One or more Input Transformations.
- One or more Data Validation expressions.
- One or more Output Transformations.
- A "Help" message describing the field to the user. This message may include executable expressions, so that the latest versions of Code Tables etc used to validate the Data Type can be displayed as part of the message.

SHAPE is supplied with a number of Standard Data Types. In addition, a site can define its own Data Types, and each application can have its own. The Standard Types supplied will include:

- All the 'ordinary' computer language Data Types, i.e. BOOLEAN, INTEGER(Width), FLOAT(Width,Decimals), COMPLEX, and CHARACTER.
- NUMERIC (any numeric data type).
- 555(etc): defined in terms of a Format by Example expression.
- YN ('Y' or 'N'), YESNO ('YES' or 'NO').
- ALIAS: must be a valid LOGOS/SHAPE ALIAS at the site.
- YYMMDD, DDMMMYYYY, etc: all the various date combinations.

2.11. Data Transformations

Data Transformations are applied on INPUT or OUTPUT of fields on a screen, to map the external representation (e.g. '08AUG87') to the internal form required by the Application (e.g. 19870808). Transformations are applied to a field as specified in the fields DATA TYPE (see above). Data Transformations are defined by:

- A Name, Description and Documentation.
- A Parameter list.
- The Result Data Type (internal or external)
- The "capacity" of the Transformation (numbers, character vectors, matrices or enclosed matriuces).
- APL code defining the transformation.

SHAPE contains a number of Standard Transformations, and as with Data Types, the site and the individual programmer can define their own. The Standard Transformations will include:

- All Transformations required to implement the Data Types described above.
- TOUPPER/TOLOWER/TRANSLATE(table): various character- by character translations. Minus signs, comma/dot, and User-Definable tables.
- Trimming of trailing / multiple blanks.
- Justification (left, right, centre).
- STRING SEARCH/REPLACE (e.g. APL Keyword/Symbol Mapping).
- User-definable APL functions.

For each Data Type, one or more transformations may be combined on input and output, to achieve the desired effect.

2.12. Data Validation

Data Validation is applied to data AFTER Input Transformation, to see if it conforms to the Data Type. Validations are defined by:

- A Name, Description and Documentation.
- A Parameter list.
- The "capacity" of the validation (numbers, character vectors, matrices or enclosed matriuces).
- The result type: (bits, line nos, cells+line nos).
- APL code defining the validation.

Regarding the "result type": SHAPE uses the result to highlight errors on the screen, and we'd like to give the programmer some freedom rather than forcing a "standard" result type which may be difficult for the inexperienced programmer to construct. The same comment applies to our allowing people to write Transformations and Validations with different capacity" for handling data. If the capacity is low (vectors), SHAPE loops around the Validation automaticaly. Same idea as "function rank" in many ways.

Standard Validations supplied with SHAPE will include:

- All Validations required to implement the Data Types described above.
- REQUIRED: not all blank.
- ALPHA: all blank or alphabetic chars.
- ALPHANUM, DIGITS, ELEMENTOF(list), etc.
- RANGE (max,min)
- TABLE(tablename): character or numeric table.
- SINGLE, DOUBLE, LENGTH(=n) or ($\leq$n): number of 'fields' allowed, where a blank is field delimiter (a la checking length of result of QUAD-VI).
- User-Defined function.

2.13. Profiles

Profiles in SHAPE are very much the same as in Viewpoint, allowing the user to define the appearance of his screen, default MACRO library, Print Destination and so on.

2.14. Macros

SHAPE Macros are the same as Viewpoint Macros. In fact, we would like to steal the code for PROFILE/MACROS direct from VIEWPOINT. A MACRO is a sequence of commands, separated by semicolons, which are to be executed in sequence when the name of the MACRO is entered on a Command Line.

2.15. Menus

A menu is a screen on which various options are displayed, and the user is encouraged to make a selection. The layout of a menu is Site-definable; a menu is in fact an ordinary screen, which must conform to a set of rules, i.e. there must be a field in which to display the options, and so on. The site decides whether menu selection is done by entering a number, a mnemonic code, pressing a PF-key, or any of the above. The options which can be selected from a menu may be Commands, Screens or new Menus.

Menus are defined in terms of:

- A Name.
- A Short Description.
- An optional text to be displayed in a window on the Menu, if such a window is part of the sites Menu-Layout.
- An entry-point into the hierarchical HELP system, to be used if the HELP command is used without identifying a particular option.
- A set of COMMANDS, SCREENS and APL OBJECTS local to this Menu.
- Some APL CODE to be executed before displaying the Menu.
- Some APL CODE to be executed on leaving the Menu.
  for EACH option on the menu:
  - Its TYPE (Command, Screen, Menu).
  - Its Name.
  - A short description, which appears on the menu.
  - An entry-point into the HELP system.
  - An access table: a list of those users, or groups of users, which may select the option.

As is suggested above, SHAPE generates a system where each MENU is a NODE on the paging file (unless asked not to).

2.16. Commands

A command is an action which can be triggered by selecting an option on a menu, by pressing a PF-key, or by entering the Command on a Command Line.

- The Command Name.
- One Abbreviation allowed for the Command Name.
- A Short Description, and a Detailed Description.
- A Short Name for the command, to appear when a PF-key is defined for the Command, and the PF-key line is crowded.
- How many arguments it takes, what type they are, and whether or not they are required.
- An Entry Point into the HELP system, to be used by the HELP command supplied with SHAPE (user types HELP command).
- Command Type (System or Screen).

SHAPE is shipped with a large number of Standard Commands, to perform all kinds of utility operations. Each Site may define a set of Public Commands, available to each application. In addition, each System, Menu or Screen can have its own Local commands.

2.17. PF-Keys

A PF-key definition describes a link between a PF-key and a Command. It is simply defined in terms of:

- The key to be pressed.
- The name of the Command to be executed.

In SHAPE, a 'PF-key' can refer to any key which releases the screen, such as CLEAR, ENTER or a PA-key, in addition to true PF-keys.

2.18. Screens

A Screen is a Loop in the Application, where the terminal displays a collection of fields. Some of the fields may be initialized to contain certain values before the loop starts. For each pass throught the loop:

1. Fields which have changed are written to the screen according to the defined Output Transformations.
2. If there is a 'SET FIELD' COMMAND in the type-ahead buffer, it is executed, otherwise:
   The keyboard is unlocked, and the user is free to do as much damage as possible, and end it by pressing a PF-key (or another key such as CLEAR or ENTER).
3. Input Transformations and Unconditional Data Validation is performed for fields which were modified by the user.
4. If any Validations failed, the fields in error are highlighted, and no COMMAND is executed.
5. Depending on which key was pressed, and the contents of the Command Line (if there is one), a COMMAND is executed, after any COMMAND-dependant Validation is performed.
6. If the COMMAND succeeds, and a Screen State transition is defined, the required changes to the Screen are made (in the worst case, the field is re-formatted).

A Screen is defined by:

- A Name.
- A Short Description, and a Long Description.
- An entry point into the HELP system to be used if the HELP command is executed with the cursor in a Field for which there is no Field-specific HELP message defined.
- A set of COMMANDS and APL OBJECTS which are local to this screen.
- Some APL code to be executed on entry into the Screen.
- Some APL code to be executed on exit from the Screen.
- A set of Fields.
- Possibly some Field Groups.
- And even a number of Field States.
- The minimum size Physical Screen on which this screen can be displayed.
- The position and absolute or relative size of this screen (relative to the Physical Screen). If this setting is anything other than $(0,0),(0,0)$, then the screen is an Overlay, i.e. any image displayed when this screen is initialized should be preserved as a "background" and this screen only fills part of the Physical Screen.

2.19. Screen States

A screen can have several States. A typical example of this is a screen for modifying records in a file. To start with, only the KEY field is open for input. Once the user has entered a valid key, the KEY field changes colour and is "LOCKED" (output only), until the record is saved again. During editing, the other fields are opened for input. Screen states are defined in terms of modifications to be made during transition to the state, from other states. Thus, a state is defined by:

- A Name.
- A Short Description and a Long Description.
- A Transition List, e.g. FLD1 becomes BLUE, REVERSE, INPUT.
- Whether or not Commands may be used, which would exit from the screen and thus lose changes made to the screen.

2.20. Fields

Fields are the windows in the screen through which the data can be seen. Fields have a large number of attributes:

- A Name (optional, and not required unless the application wishes to refer to the field, e.g. for fixed-test fields).
- Data Type, or optionally Input, Output Transformations, and Validation(?).
- Position (absolute, relative to a boundary of the screen, or relative to an edge of another field).
- Size/Shape, or position of bottom right corner (absolute or relative, as above, or defined by the size of the data, at run-time).
- Type (Input, Output or Graphics).
- Other screen attributes: Intensity, Colour, Skip to next Input Field, How to write Trailing Blanks, and to read Trailing Nulls).
- Initialization Text.
- Field Group(s) to which the Field belongs.
- Which Screen States the Field is present in.
- Entry Point into the Help system.

2.21. Field Groups

Field Groups allow several fields to be manipulated as a group. A Field group is defined as follows:

- A Name.
- A Short, and a Long Description.
- The Fields which are members of the Group.
- Whether the Fields in the Group SCROLL, either Verticaly or Horizontaly.
- In addition, a Field Group CAN HAVE settings for all the attributes of a Field (with the exception of Name, Position and Field Groups membership), so that Fields which are Members of the Group can be forced to have certain attributes (typically, this would be used to specify Groups of Fields which are present in different Screen States).
- Whether or not the Field Group is a "Table", and if it is a Table, the size of the region which is reserved for the Table, on the Screen.

2.22. Field Tables

A Field Table is a special Field Group, which is used for displaying Data which is in tabluar form (Records by Fields). A Field Table may be WIDER than the Screen which it is a part of, and is used together with the Standard Commands "RIGHT" and "LEFT", "HOLD" (and others) which move fields on and off the region reserved for the Table, on the Screen.

2.23. Special Fields

SHAPE recognizes several "Special Fields" which have special meaning and/or are updated automaticaly by SHAPE:

Field Name	Description
/COMMAND	The COMMAND line
/MESSAGE	A MESSSAGE line used to output one-line error messages, receipts and so on.
/SYSTEM	A line on which the name of the System is centered within horizontal box-drawing chars.
/SCREEN	Name of Screen, nicely centered.
/STATE	Name of active Screen State, centered.
/SCREENSTATE	Names of Screen and State, as above.
/PFKEY	A PF-key line.
/ALIAS	The ALIAS of the user.
/TIME	Time of Day, formatted the way the Site wants it.
/DATE	Date (as above).
/ETC	We'll think of some more.

2.24. Graphs and Images

Not defined at this point. See chapter 12.

2.25. Helps

Not defined yet. See chapter 11.

3. Welcome to SHAPE

The remainder of this Document is a mixture of Screens we have imagined (to give you some idea of what we want SHAPE to look like), together with comments which expand on the Definitions if Chapter 2. The form will be somewhat like a Users Manual, except that the intended audience will not be "New, Naive users of SHAPE"! Here, to start you off, is the SHAPE "Main Menu":

```
---------- System:  GITTE.                    ----------

---------- Shape 0.0 ----------

                    WELCOME TO SHAPE

   DOCUMENT   Produce Documentation.     DATA     Define DATA types.

   SYSTEMS    System Definitions.        FILES    Describe FILES/DATASETS.

   MENUS      Define Menus.              SCREENS  Create Screens.

   PROFILE    Edit User Profile.         APL      Maintain APL-code.

   MACROS     User-defined Macros.       ADMIN    System Administrator.

              To select an option, use the tab key to move the
              cursor opposite the option name, or type the name
              of the option on the Command Line, and press ENTER.

---------- 1=Help, 3=Exit ----------
```

Screen 1: SHAPE Main Menu

As some of you will be able to see, it looks a lot like VIEWPOINT. The only difference (apart from the options available) is that SHAPE (like LOGOS) believes in ALIASes rather than User-Numbers!

4. Systems, Phases Variants and Environments

This Chapter describes the screens used to edit the definitions of Systems, Phases, Variants and Environments.

4.1. Systems

The LOGOS pathnames used to store a SHAPE application are of the form:

ALIAS.SYSTEM.[VARIANT.PHASE.]MENU1..MENUn.*?

Thus, the generation of End Environments is a question of permuting multiple working directories while building the Page Files used to run the system.

Select the SYSTEMS option on the Main Menu. The following Screen will appear:

```
---------- System GITTE.                                    ----------

System   1 of  2                                  Systems in Library: GITTE
---------------------------------- SYSTEMS -----------------------------------

  >System            >Description / Documentation     >Pha  >Var  >Env  >Shr
   GITTE.CTS          Clinical Trials System            3     2     3   YES
   GITTE.INVOICE      Invoice Control System            1     1     1   NO

---- 1=Help, 2=Sned, 3=Exit, 4=Add, 5=Delete, 6=Zoom, 7=Up, 8=Down, 9=Save ----
```

Screen 2: The Systems Screen

As VIEWPOINT does, we aim to make good use of the ZOOM concept. In the above Screen, one can ZOOM on the Description field to to see the System Documentation, on the Phase/Variant/Environment Fields to define any of these, and on the Shared field to set Developer and User Access. Zoom on the System field itself brings up the System Configuration Screen.

If NONE of the Systems on an account have more than one Phase, Variant or Environment, then these fields will be removed from the screen. The user will have to enter a Command the first time he wishes to create a multi-phase or multi-variant system.

Tab to the first line of the Systems field, and ZOOM:

```
---------------------- System GITTE.CTS: Clinical Trials --------------------

---------------------------- System Configuration ----------------------------

   Compilation                 Decomment   Diamondize     Lock      Secure
        For Prod Environs        YES         YES          YES       YES
        For all others           NO          NO           NO        NO

   Strict Production System: YES     Use Paging Files.....: YES
   Translatable Message Prefix . . . . . . . . . . . . . . . . .: CTSMSG
   No. of Back Versions to keep:  SYS   10, FILES  10, MENUS   5,
                                  DATA  10, APL    10, HELPS   3.

-- File Declarations [0] ------------------------------ File   1 of   10 -----

   Symbolic Name           TYPE  >Description
   DIR                     APL    CTS Directory
   DATA                    APL    CTS Data File
   UPDATE                  APL    CTS Update Request Log
   SYS.TABLES.CURRENCY     APL    This Years Standard Currency Rates
   SYS.TABLES.CUSTOMERS    VSAM   The Customer Register

------------ 1=Hlp, 2=Snd, 3=Back, 4=Add, 5=Del, 7=Up, 8=Dn, 9=Sav ------------
```

Screen 3: System Configuration

Note the [0] after "File Declarations". This is the LOGOS version number of the variable which contains the File Declaration. Wherever such a version number is indicated, the Command 'VERSION n' will allow the user to flip back and forth between the current version, and previous versions (as far as LOGOS keeps them). The SAVE Command will save whichever version in on the Screen at the time, as the new current version. A "HISTORY" command displays an overlay with the results of LOGOS LIST +V.

Once "Strict Control" has been selected, it cannot be withdrawn, except by the System Administrator. The same goes for all related settings. For example: once an Environment has been declared to be a PROD environment, this cannot be changed IF Strict Control is specified. The Site Administrator can force Strict Control upon any System, if he desires.

The "Translatable Message Prefix" is a prefix to be used for all variables or functions which are Translatable. In other words, only objects with these prefixes should be stored in the directories for variants other than the "Base" Variant. This makes it easy to identify objects which should be (should have been) translated. There will be reports to display these messages, and screens from which to edit several Variants at once. Alternatively, this "prefix" might be a "regular expression", but this might frighten the naive user.

This screen allows you to specify six retention limits (number of backup copies to keep).

4.2. Phases and Variants

Press PF3 (Back), Tab to the first line of the Phases field on the Systems Screen, and Zoom. Alternatively, type 'BACK;ZOOM PHASES(1)' on the Command Line and press Enter. The following screen will appear:

```
---------------------- System GITTE.CTS: Clinical Trials --------------------

---------------------------------- Phases ---------- Phase    1 of    3 -----

      Name       >Description                                >Shared   >Prod
      TEST        For DEVELOPMENT                             YES      NO
      PROD        PRODUCTION version                          NO       YES
      TEST2       NEW INPUT FUNCTIONS                         YES      NO



---------------------------------- Variants --------- Variant   1 of    2 -----

      Name       >Description                                >Shared   >Base
      ENGLISH     BASE                                        YES      YES
      DANSK       DANISH LANGUAGE VERSION                     YES      NO



---- 1=Hlp, 2=Snd, 3=Exit, 4=Add, 5=Del, 7=Up, 8=Dn, 9=Sav, 11=Env, 12=Shr ----
```

Screen 4: Phases and Variants

The cursor is in the Phases section of the Screen (it would have been in the Variants section if you has Zoomed on them). From this screen, you can add, delete or change Phases or Variants, and their documentation. Zooming on the "Shared" field takes you to the same Access Screen as you would get to from the Systems Screen (see Screen 6).

The "Prod" Field indicates whether or not a Phase is included in any Production Environments. Depending on Site and System parameters, this may place constraints on changes to such Phases.

4.3. Environments

Pressing PF11 (Environments) takes you to the same screen as you would have arrived at, had you Zoomed on the Environments field in the Systems Screen:

```
---------------------- System GITTE.CTS: Clinical Trials ----------------------

------------------------- End Environments ------ Environment    1 of   4 ----

 Name      Pro  >Overlaid Phases          Description                      >Shr
 CTS       YES   PROD                     PRODUCTION                        NO
 CTST      NO    PROD,TEST                The TEST environment              YES
 CTS2      NO    TEST2,PROD               NEW INPUT FUNCTIONS               YES
 CTS3      NO    TEST2,TEST,PROD          NEW INPUT FNS, AND TEST STUFF     NO

--- 1=Hlp, 2=Snd, 3=Ex, 4=Add, 5=Del, 6=Zm 7=Up, 8=Dn, 9=Sv, 10=P+V, 12=Shr ---
```

Screen 5: End Environments

Environments are defined as Overlaid Phases. For example, The normal TEST Environment called CTST is PROD overlaid with TEST. Each Environment is generated with all Variants.

The Command GENERATE (couldn't find a PFkey for it) builds the selected Environment. For Production Environments, the Generation job enters a queue where it is held until the System Administrator releases it (if the Site desires).

Whether or not it will be necessary to GENERATE applications should possibly be discussed further. It is possible that SHAPE should use its own 'Paging Files' rather than using overly general LOGOS utilities. SHAPE has knowledge of the structure of an application which should make it possible to bring code in more intelligently than LOGOS could, and also to insert edited (and recompiled) objects straight into the paging file after they are edited. On the other hand, LOGOS 'audit files' and the 'distribute' command may be a viable alternative. We need more input on this.

4.4. Access Control

There are many routes to the Access Screen from the screens described earlier in this Chapter. Once you get there, it looks like this:

```
---------------------- System GITTE.CTS: Clinical Trials ---------------------

------- Developer Access [0] ----------------------- Entry     1 of     3 -----

        Alias           Access          Phases              Variants
        DANB            .WRX            TEST,TEST2          ENGLISH
        GITTE           CWRX            ALL                 ALL
        MKROM           ..RX            TEST*               DANSK



------- User Access [0] --------------------------- Entry      1 of     4 -----

        Alias/User      Environment Mask                    >All Menus?
        /ALL            CTS                                  NO
        DANB            CTS2,CTS3                            YES
        GITTE           ALL                                  YES
        2848199         CTS*                                 NO



-------- 1=Help, 2=Send, 3=Exit, 4=Add, 5=Delete, 7=Up, 8=Down, 9=Save --------
```

Screen 6: Access

Until APL maintains an ALIAS list for all users, we will have to allow user-numbers in the user-access table. Note that the use of ALIASes in the User Access means that the we must somehow be able to map a LOGOS ALIAS to a user-number.

"All Menus" zooms to a screen which builds a Menu Structure tree, and makes it possible to add or remove access to menus for a given user (see next page).

The "alias" /ALL gives access to all users. An alternative is /ACF which gives control to the sites "Access Control Facility". This is a set of site exit routines which can be modified by a site to talk to their Access Control Facility (e.g. RACF) via SACF. The default versions of these functions look at an APL file which can be maintained from the System Administrator number. This allows central control of access to applications, either through maitenance of the APL file, or by declaring SHAPE applications as "resources"

to be protected by the external ACF.

```
---------------------- System GITTE.CTS: Clinical Trials --------------------


------------------------------ Set Menu Access -------------------------------
  For user.: GITTE          Environments...: CTS          Access...: BLUE
  Root Menu: MAIN                                         No Access: TURQUOISE

  Point Cursor at MENU, press PF4 (add) or PF5 (del) to alter access to MENU
------------------------------------------------------------------------------

     Main---+---1.Update---+---1.1.Kits
            !               +---1.2.Status
            !               +---1.3.Finance
            !
            +---2.Reports--+---2.1.Kits
            !              +---2.2.Status
            !              +---2.3.Ad Hoc
            !
            +---3.Steward--+---3.1.Delete  Trial
                           +---3.2.Currency Rates
                           +---3.3.Update(*)


----- 1=Hlp, 2=Snd, 3=Back, 4=Add, 5=Del, 6=View Menu, 7=Up, 8=Dn, 9=Sav ------
```

Screen 7: Edit Access to Menus

This Screen allows the setting of access to individual Menu options, while the user browses through the Menu structure. Access may be set for a single user at a time, or for a group of users. To change access, the cursor is pointed at an option, and PF4 is pressed to ADD access, PF5 to DELETE it. PF6 (Zoom or View Menu) displays the MENU pointed at, AS IT WOULD APPEAR to the user in question.

Menu Options to which the user does not have access can be highlighted in a different colour (or blinking, reverse etc), or optionally removed from the display.

5. Data Types, Transforms, Validation

Data Types are the basic building blocks of SHAPE Applications. Fields in files and on screens are defined by their Data Type.

A Data Type is a declaration of an optional Internal Data Type (only required if you wish to make use of the Data Type in a File Definition), plus the Input+Output Transformations which are used to convert the data between its external and internal representations.

A Data Type may have Validations - which check that the data is correct for the Data Type. Input Transformations may also perform validation: the final stages of converting to the internal represen- tation often require validation, and it would be a pity to force the user to do this validation twice.

A Data Type may have parameters, where simple variations of the Data Type are required. For example, the Standard Data Type "Float" has two parameters: & WIDTH and & DECIMALS. Typicaly, these parameters are passed on to Transformations and Validations within the Data Type.

A new Data Type can be built upon an existing one, for example one might create a Data Type based on the 'Standard' Data Type called "DDMMMYY', with constraints on the dates.

The naive user may find the definition of Data Types and Validations a little heavy going, and Transformations even worse. However, we hope that we can supply enough Standard Data Types that the user should not need to use these facilities unless he wants to do things that only programmers can be expected to do!

5.1. Data Types

Return to the Main Menu and tab to the DATA option, or enter the "DATA' Command from any other screen (if you have made any modifications you will be prompted to save them before continuing to the Data Screen:

```
---------------------- System GITTE.CTS: Clinical Trials --------------------


--------------------------------- Data Types ------------ Type   1  of   4 --

  >Type         >Parameters          >Description
   TID                                Trial ID (XXX/YYY/nnn/ZZZ)
   KEY                                Trial KEY: RANGE(1000,9999)
   KITCODE                            Kit Code Number: RANGE(100,99999)
   DATEAFTER     &DAY1                DDMMMYY AND .GE &DAY1

...

- 1=Hlp, 2=Snd, 3=Bck, 4=Add, 5=Del, 6=Zm, 7=Up, 8=Dn, 9=Sav, 10=Trn, 11=Val --
```

Screen 8: Data Types

The Data Types Screen allows the user to maintain a set of Data Types for the application. The actual definitions of each Data Type are accessed by Zooming on the Type field. A lengthier description may be viewed/edited by zooming on Description. PF-keys

10 (Transforms) and 11 (Validation) take you to a screen similar to this one, from which Transforms and Validations may be edited.

To take a closer look at one Data Type, Type ZOOM TYPE=DATEAFTER, and press Enter:

```
---------------------- System GITTE.CTS: Clinical Trials --------------------


------------------------ Edit Data Type: DATEAFTER [0] ----------------------
  >Type         >Parameters          >Description
   DATEAFTER     &DAY1                DDMMMYY AND .GE &DAY1

  >Base Data Type: DDMMMYY            Internal Data Type: I8
                                      Screen Field Width: 7
---- >Input Transforms (Before) --------------------------- Line   1 of   0 ---


---- >Validation (After)  ---------------------------------- Line   1 of   1 ---
      .GE &DAY1

---- >Output Transforms (After)  --------------------------- Line   1 of   0 ---


---- Validation Example. Parameters: &DAY1=19870101 --------------------------
Environment:
Input...: 01AUG
Internal: 19870801
Output..: 01AUG87
-- 1=Hlp, 2=Snd, 3=Back, 4=Add, 5=Del, 6=Zoom, 7=Up, 8=Dn, 9=Sav, 10=Expand ---
```

Screen 9: Edit Data Type

This screen allows definition of a Data Type. Zooms on the Transforms or Validation fields take you to the screens for viewing or editing these, to allow definition on the fly when required. The "Validation Example" fields at the bottom of the screen allow the user to TEST his data type immediately, having samples validated, seeing the internal format and the display after Output Transforms, too. The "Environment" field allows you to specify a Menu Number or Name, so that objects required to execute the Transformations or Validations are paged in.

The "Expand" Command (PF10) copies the definition of the Base Data type, so it can be modified rather than built upon. By default, any additional Input Transforms are performed BEFORE those of the Base Type. Validation and Output transforms are performed AFTER those of the Base. This can be changed simply by overwriting the "Before" or "After" texts on the screen.

Perhaps it should be possible to set a "Default Value" for a Data Type? For example, for dates it might be nice to say default is current date). This can of course be done by via Input Transform Or in the Initialization code for the Screen in question.

5.2. Transformations

Going back and selecting the Transforms or Validation screens displays screens more or less identical to the Data Types screen. If one zooms on a particular Transform, the following screen appears:

```
---------------------- System GITTE.CTS: Clinical Trials --------------------


------------------------ Edit Transform: TRANSLATE [0] ----------------------

  >Name          >Parameters         >Description
   TRANSLATE      &TABLE              Translate according to table

-- APL Code ------------------------------------------------ Line   1 of  7 ---

   #goto simple #if #simple DATA
   t #is #ravel #with #disclose DATA
   result #is #ravel -1 -1 0 0 FmtQps t
   mask[#plus #scan #ravel #shape #on #disclose t] #is 1
   result #is &TABLE[#quadAV #iota result]
   result #is m 1 #cut #enclose result
   result #is (#shape #with #disclose DATA) #reshape #with #disclose result
   #exit
---- Transformation Example. Parameters: &TABLE=SiteUppertran ----------------
Environment:
DATA...: En prø ve: æ ø å
Result.: EN PRØ VE: Æ Ø Å
----- 1=Hlp, 2=Snd, 3=Back, 4=Add, 5=Del, 7=Up, 8=Dn, 9=Sav 12=run ------------
```

Screen 10: Defining a Transform

A transform is an APL function, but we do not ask the user to enter a function header. We will give the function some internal name, localize variables automaticaly, and call it when we need it. The author of the code for the Transform specifies whether it is capable of handling a vector of numbers, or characters: one line, a matrix (multi-row field) or several fields at once (as a vector of enclosed matrices). The example above can handle the character argument types. If a transform can handle numbers, SHAPE will possibly pass values from many fields to it as one numeric vector.

If a Transform wishes to indicate errors in the data, it can call a 'SIGNAL' function with an indication of the lines (and if necessary the cells) which are in error, and an optional argument which is the error message to be displayed (overriding the default).

The Transformation function will be tested for its ability to handle all the forms of data which the user has declared that it can, when PF12 (RUN) is pressed.

5.3. Validations

Editing Validations is done one a screen which is almost identical to the Transformations screen:

```
---------------------- System GITTE.CTS: Clinical Trials --------------------


------------------------- Edit Validation: RANGE [0] -------------------------

  >Name          >Parameters          >Description
   RANGE          &LOW,&HIGH           Are numbers between &LOW and &HIGH?

   Defined on Numbers, Lines, Matrices or Fields.....: N
   Result type: Bit Mask, Enclosed Line Nos or Cell+Line Matrix: B
   Error Message: Entered Data not in Range
-- APL Code ------------------------------------------- Line   1 of  7 ---

   result #is (DATA #ge &LOW) #and DATA #le &HIGH




---- Validation Example. Parameters: &LOW=100, &HIGH=200 ---------------------
Environment:
DATA...: 100 200 300
Result.: 1 1 0
------------ 1=Hlp, 2=Snd, 3=Back, 4=Add, 5=Del, 7=Up, 8=Dn, 9=Sav ------------
```

Screen 11: Defining a Validation

A Validation is also an APL function, which returns a result telling whether the arguments were valid. There are several ways to return the result. The example above is the simplest possible form: a numeric vector argument, and a Bit Mask result. SHAPE will combine field values and decode the results as it sees fit, in the name of efficiency. For example, the above function would only be called once, no matter how many fields there were on a screen requiring that validation, or what shapes these fields have.

6. Files

Most APL applications use files of one kind or another. SHAPE believes in three types of files:

1. "Simple" file systems:
   - OS/370 Sequential Datasets, accessed through TSIO (AP370).
   - "Flat" APL files (or parts of APL files), accessed via APL file primitives.
   - VSAM sequential or "key sequenced" Datasets, which we use VSDI (AP123) to get at.
2. Data Base Managers:
   - IBM's Data Base 2 (DB2), accessed through SSQL (AP127).
   - VIEWPOINT, accessed via VPUTIL.
   - perhaps other, like ADABAS.
3. "Raw" APL Files:
   - Free-form APL files, structured any way the programmer pleases. Access is of course through APL file primitives.

6.1. File Handling

SHAPE attempts to make it possible for the programmer to handle all files of type 1+2 (the record-oriented file types) in the same manner. Thus, the user can declare "record layouts" using the same terms (field types, etc) for all file types (and indeed the same data types as used to define fields during screen design). Selection of records is done using one or more "key fields", or using a Query Language (Viewpoint Constraints) for all file types. SHAPE utilizes the features of the underlying file systems wherever possible. Thus, handling O/S Datasets may reuire reads and writes of the entire Datasets before and after use, while VSAM files may be handled much more efficiently, as long as all selection of records is done using the VSAM "Key" fields.

SHAPE will also contain utility functions which allow the knowledgeable programmer to use features of the more advanced file systems directly, if he so wishes. For example, it might be more natural to extract data from DB2 using SQL statements, rather than giving SHAPE a Viewpoint Constraint. The first thing SHAPE does is translate the Constraint to an SQL statement, anyway.

There will also be a complete set of utilities for handling Free-form APL files. It is important that there are SHAPE cover-functions for ALL file operations, and that the utilities allow references to files to be the "symbolic name" declared in the system configuration screen, rather than the actual physical file name. In test/production environments, the cover-functions automaticaly select which files to use. In "Strict" production systems, direct file access should be prohibited. A utility will be provided for the Systems Administrator so that he can verify that applications do not make unathorized use of QUAD-TIE, or QUAD-SVO. Legitimate uses of QUAD-SVO, for example for talking to server tasks, will have to be documented.

6.2. Interfacing Files to Screens

SHAPE will support two simple ways to connect a screen to a file, corresponding (more or less) to the "list" and "form" modes of the Viewpoint RECORDS module. These interfaces consist of:

1. A command in the "Screen Design" module of SHAPE, allowing automatic definition of fields by copying them from a File Definition. For "list" mode, a "Field Table" can be created as well. The inverse operation will be available as a command during FIle Design.
2. A set of utility functions (Commands) which can be performed when the user presses PF-keys, to move data between the screen and the files.

Using these interfacing tools, the user can create the following two types of screens in a very short time:

1. A screen which has a "key" field. The user fills in the key, and presses a "read" PF-key. The corresponding record is fetched, and the user is free to update the data. Another PF-key saves the data again. Othe PF-keys allow the user to add, delete or print records.
2. A screen with a "constraint" field, and a "Field Table" which is a window on the records selected from the file. If there is more data than there is room for on the screen, the user can scroll horizontaly or verticaly. This screen should support as many as possible of the Viewpoint "Records" commands such as ADD, DELETE, SHOW, HOLD, SEND and so on, and might not make it into release 1.

Both of the above screen types can be modified to contain other information besides the actual file data, as long as certain rules regarding naming of fields and so on are followed. A third screen type which combines the two above, and contains a "Header" record from one file, together with several "Body" records linked to it from another file, might also be useful (it certainly seems to be a common occurrence in APL applications systems).

6.3. File Design

The following screen allows you to add, change or delete file declarations. Zoom on the "Description" to edit the files documentation. Zooming on the "Shr" field lets you edit access to the file declaration (not to the files themselves). Zoom on "File" to look at the details of the file declaration; the record layout, external file names, and so on:

```
---------------------- System GITTE.CTS: Clinical Trials --------------------

-- File     1 of   5 --------------- Files ------------ In Library GITTE -----

 >File                 Type     >Shr   >Description
  CTS.DIR              APLFREE   YES    CTS Directory
  CTS.DATA             APLFREE   YES    CTS Data File
  CTS.UPDATE           APLFREE   YES    CTS Update Request Log
  INVOICE.INVOICES     VSAM      -      Invoice file shared with CICS COBOL
  DB2.EMPDATA          DB2       YES    IBM's DB2 example data set.

-------- 1=Help 2=Send 3=Back 4=Add 5=Delete 6=Zoom 7=Up 8=Down 9=Save --------
```

Screen 12: Files

Lets ZOOM on the "INVOICE" file, to take a close look at how we define a Record Layout:

```
---------------------- System GITTE.CTS: Clinical Trials --------------------

-------- File INVOICE.INVOICES(VSAM) - Invoice file shared with CICS COBOL ----

             External File Names                        Update?
        PROD: SYS1.VSAM.INVOICES                        YES
        TEST: DUMMY                                     YES

-------- Record Layout ------------------------------------ Field 1 of 3 ----

   Name          Type       Internal  Xlate    Start Width Dummy
   CUSTNO        CUSTNO     INTEGER   PACKED       1     4 1000+?8999
   DATE          YYYYMMDD   INTEGER   EBCDIC       5     8 19870101
   DESC          CHARACTER  CHARACTER EBCDIC      13    40 ' '

-------- 1=Help 2=Send 3=Back 4=Add 5=Delete 6=Zoom 7=Up 8=Down 9=Save --------
```

Screen 13: File Design

For "Free APL" files, the bottom window is used to display the file documentation, rather than a record layout. For "Flat APL" files, the top window contains fields to let you define a component range on the file to be used to store the data, and perhaps how you want to store it (as real flat character matrices, or cut and enclosed along one or more dimension).

The "Dummy" field allows you to define what kind of data you want to see if you read from a "DUMMY" file. The "TEST" version of this file has been declared as "DUMMY". This means that while testing the application, file reads will generate random data, as specified on this screen. Writes to DUMMY files go up in smoke (or perhaps we will let them live for the duration of the session so they can be re-read?).

The "Update" field following the external file name allows some more control during debugging (see chapter 13 for more information). You may specify that a file is not to be updated, or that updates are to be "verified", i.e. displayed in a fullscreen editor before being performed, so that small bugs in update routines can be corrected.

6.4. APL File Utilities

For the programmer who deals with "Raw" APL Component files, SHAPE contains a toolbox for doing housekeeping. The housekeeping tools include a screen from which the programmer can document his APL files, browse through them, copy all or part of one file to another, compare files, manipulate access matrices and look at timestamps. The command UTILS in the FILES screen brings up this utility screen:

```
---------------------- System GITTE.CTS: Clinical Trials --------------------

File  1 of 1          File Name prefix:                  File Library: MKROM
------------------------------ APL File Utilities -----------------------------
                                        ---comps---   ---bytes---
  Command     >Name                    >Shr >1st  >Last   Used   Rsvd  >Date
  copy 1-10   S0123451524               NO     1    100   594K   1.1M  1987-08-05
  to          SCOPY

-------- 1=Help 2=Send 3=Back 4=Add 5=Delete 6=Zoom 7=Up 8=Down 12=Run ---------
```

Screen 14: File Design

From this screen, the user can issue commands such as COPY, PRINT, BROWSE, SEARCH, COMPARE, DELETE. Zoom on the "Shr" field brings up the access matrix for fullscreen editing. Changing the 1st or Last component numbers drops or pads (when you press PF12). Changing the Size entries resizes the file. You can zoom on the file name to look at documentation, or on the Date (which displays the timestamp of the first component) to look for the most recent changes to the file. Zoom on the 1st and Last component fields initiates BROWSE at the start or the end of the file.

A similar screen could be created to manipulate the workspace library, but it is probably not wise to give people power to "play" with workspaces.

7. Menus.

The use of Menus for building hierarchially structured applications is widespread. Menus split applications up, usually in some logical way, and are supposed to help the user find the module which he wishes to run. Used sensibly, menus can be valuable tools for the programmer as well, making it possible to group modules which depend on the same code or date under one branch of the menu tree. The programmer can specify that the required code and data be brought into the workspace at one point, rather than doing so several times.

7.1. How does SHAPE use Menus?

SHAPE tries to help programmers implement systems as described above. If the application is large or complex enough to need it, SHAPE will use the Menu structure to page modules in and out of the workspace. Therefore, a Menu in SHAPE is not merely a table of Options and Commands to be executed for each Option, but also a point in the application where code "entry" and "exit" code segments can be executed, and modules can be brought into the workspace. If a menu has any "local" objects, the menu and its associated objects will be a NODE in a LOGOS paging file.

7.2. Shortcuts

Although tree-structures are nice and help you find your way around, and help the programmer split the application into modules, the experienced user quickly gets tired of climbing up and down branches of trees to get from one module to another. Sometimes he needs to take a quick look at the currency rates which he can see in another screen, before filling in the last item on the screen which he has already spent half an hour updating. We can think of three types of "shortcuts" which are useful. You want to be able to activate another module in three different ways:

1. Forget what you are doing now, and go to the new module (but remind the user to save changes first, if he has made any).
   We will provide a Utility Function to go from here to anywhere else (after checking the no-exit bit).
2. Call up the new module temporarily, preserving the current environment intact until we return to it.
   There will be a function which checks how much the new+old screen have in common (how "far out" their menu paths diverge), and page in the required modules, execute initialization code, call the new module, and return (this will not necessarily work, unless the new module is reasonably well-behaved).
3. Start up another module and allow the user to flip between the new and old modules (perhaps displaying part of each session on a split-screen).
   Multi-session SHAPE will definitely not make it into release 1. However, we have the technology (VUPT AUTOPILOT, NWS) to do anything we want to in this area. Multi-session IDSH may beat us to it, and many sites (all three in Denmark) are already running TPX, MASS or other VTAM multi-session handlers, so there may not be much demand for it. However, having played a little with IBM's ISPF we have found split-screen mode to be very useful. Split-screen raises some interesting questions with respect to how "dynamic screens" with variable-sized scroll fields (etc) should react to having the screen size changed in mid-session, but if ISPF can do this, so can we!

There is no reason to stop at two parallel sessions, or to limit ourselves to modules within the same application.

There are quite a few potential problems with badly-behaved modules wreaking havoc, especially under point 2 above. Possibly, modules will have to be marked to tell SHAPE

whether they 1) can take an interruption and 2) whether they are well-behaved enough to be called by another module.

Although we do not implement multi-session SHAPE for a while, we may wish to make a few sidekick-like features available, such as a Notepad, Calculator, Calendar and Diary, etc available from 'anywhere' in a SHAPE-controlled application, as overlays perhaps.

7.3. Defining Menus

Creating Menus is done from an "Overview" screen, in the same way as Systems, Files etc. To define the contents of a Menu, one uses the ZOOM command to get to the following screen:

```
----------------- System GITTE.CTS: Clinical Trails System -----------------

------------------------------ Define Menu ---------------------------------
                                                        Local Objects
  >Menu            >Description                      >Scrn >Cmds  >APL >Mods
   MAIN             CTS - MAIN MENU                    0     0     1     0

Entry: INITCTS
Exit.:
---------------------- Options ---------------------------- Option  1 of  1
  N    Type  >Definition          Description                    >Acc  >Hlp
  1      M    STEW                Perform STEWARD Functions      YES   YES
  2      M    UPDATE              UPDATE Data Base               NO    YES
  3      M    REPORTS             Produce REPORTS                NO    YES




- 1=Help 2=Send 3=Exit 4=Add 5=Delete 6=Zoom 7=Up 8=Down 9=Save 10=View Menu -
```

Screen 15: Define Menu

There are Three types of options: Menus, Screens and Commands. The 'Acc' and 'Hlp' fields tell whether Access control is in effect for the Option, and whether a Help message has been entered. The four numbers at the top of the screen show the number of local Screens, Commands, APL functions, and APL Code "Modules" which are Local to the MENU.

7.4. Layout of Menu Screens

Each site has its own ideas regarding the ideal layout for Menu Screens. We would like each site to be able to define a Template to be used as a base for a Menu. The system administrator creates this template by:

1. Designing a screen, on which there MUST be a "special field" called /MENU, in which the Options will be displayed at run-time. The Menu Driver will recognize two more fields: /SELECT, into which the user enter his selection, and /INFO, in which the system displays help messages.
   The screen may contain all standard field like /COMMAND, /MESSAGE, and site-defined fields to display the Date and Time, the users Initials, Telephone number or what have you.

2. In addition, the Administrator selects a number of options which control the way the Menu Driver handles the Menu Screen:

- Whether selection will be by PF-key (PF-*n* selects option *n*), by tabulation (tab field inserted), by entering an 'X' opposite the option, or by entering the number of the option in a /SELECT field.

- Whether options may be displayed in more than one column (as in the VIEWPOINT main menu), space permitting.

The Default Menu Layout gives menus like the VIEWPOINT main Menu.

8. Commands

Commands are units of work within SHAPE. Typically, when a user presses a PF-key, a Command is executed. Many screen driver systems link PF-keys directly to some code to be executed, but SHAPE defines Commands as separate entities. Commands may be entered on a Command line. PF-keys may be programmed to insert the name of a Commands into the Command Buffer. This does not mean that all Screens must have a Command line - SHAPE can manipulate the Command Buffer without having a Line on which to display it.

8.1. Command Arguments

Commands can take **arguments**. Arguments are normally supplied on the Command Line immediately following the name of the Command, but many Arguments can be "given" by the position of the cursor. The following screen is used to define Commands and their arguments:

```
------------------------- Edit Command: PROCESS [0] -------------------------

   Name      Abbr    PF-name    PF-sn  >Description
   PROCESS   PRO     Process    Pro     Process Table Entry

   Default: ACTIVE
-- Arguments ------------------------------------------------ Arg    1 of  1 ---
   Name              Data Type           Comments
   &WHAT             TABSELECT           Select Table Element

-- APL Code ------------------------------------------------- Line   1 of  7 ---

   CTSPROCESS &WHAT




------------ 1=Hlp, 2=Snd, 3=Back, 4=Add, 5=Del, 7=Up, 8=Dn, 9=Sav ------------
```

Screen 16: Command Definition

Parameters have a Data Type, and are thus subject to Input Transformations in the same way as Data in a Field on the Screen. The Transformations can supply default values for parameters, for example by examining the position of the cursor. `TABSELECT` is a Data Type supplied with SHAPE, which is a character vector which should have the form `'TABLENAME index'` or `'TABLENAME IF field=value'`. If empty, the 'index' parameter is deduced from the postion of the cursor, which must be in a Field Table on the screen. If there is only one Field Table on the screen, then `TABLENAME` need not be supplied (this is the case with most screens). For example, if there is a Field Table called 'RECORD' on the screen, the user can type `PROCESS RECORD 11` or `PROCESS RECORD IF NAME=ZORBA`, or point at a record and type `PROCESS` with no argument (or press a PF-key, if one has been defined).

SHAPE contains several "Argument Data Types" which can be used for Table Selection, and substitute some value calculated from the table and the cursors position within it, for example `LEFTMOST` which returns the data in the leftmost field on the screen, `XSELECT` which assumes the existence of a field called 'XSELECT', and returns the indices of any non-blank rows, or the position of the cursor, if all rows are blank.

8.2. Commands and Screens

For commands which are intended to work within a particular screen (as opposed to general-purpose commands which can be used from many screens or menus even), there are a couple of extra issues.

1. Although we specify validation for each field in SHAPE screens, may be fields which only have to be validated when certain commands are executed. There may also be validation expressions involving several fields, which must be performed before commands are run.
2. The successful completion of a Command may cause a Screen "State Transition".

Although these could be seen as attributes of a Command, they are more dependant on the particular screen in question, and the definitions or declarations are therefore made during Screen Design.

8.3. The Scope of Commands

If a Command is local to a Menu, this means that the APL objects required to run the Command are present in the workspace when that Menu is entered, and while all Screens and Menus "below" that menu are active. If the Command was specified as being "Active by Default", this means that it will be possible for users to call it from any of these Screens and Menus as well, unless a Menu or Screen Definition explicitly excludes it. Most Commands will probably not be "pervasive" but be Inactive until a Screen or Menu wishing to allow use of the Command declares it.

9. APL Code

In addition to writing APL code to define Data Transformations, Validation, Commands and Menu or Screen Entry+Exit expressions, the user is of course free to define his own APL functions and variables which can be called by Commands, Transformations, and so on. The user may also use "public" utility functions, or code belonging to other applications.

9.1. Modules

APL Code is stored in Modules (Logos Directories). A Menu is a special-case of a Module, and may thus "contain" APL Code. In many (perhaps most) applications, all the code will reside in MENU Modules.

The objects which are defined as "local" to a MENU may or may not actualy be stored within the Menu Module. By ZOOMing on one of the four numeric fields at the Top Right of a Menu Definition screen, the user can maintain the list of Screens, Commands, APL Objects or entire Modules which are to be brought into the workspace when the Menu runs. Each of these definitions may either be a reference to a "real" object stored within the Menu Module, or a "link" to another module.

One module may contain other modules, to any level of nesting. When a module is brought in, SHAPE assumes that objects at all levels of nesting within the module are to be brought in. SHAPE will contain utilities for finding, searching, listing and editing code stored in these modules. These utilities will be simple fullscreen front-ends to LOGOS commands such list, display, edit, etc.

9.2. Reference to Screen Data

All other programming environments we have seen (including IBM APE) map fields on the screen to a variable in the workspace. This has the advantage of simplicity from the users point of view, but leads to localization problems, symbol-table problems (probably not an issue?), and usualy requires a lot of execute statements to move data back and forth.

We have not decided how to tackle this problem. We feel that the users programmes should easily be able to access data from the screen, both int the internal form (after transformation and validation), and in the original form (users input). Whether we should store these in oodles of variables, or in enclosed arrays, is yet to be decided. If we do not use variables, should we allow the user to write programmes which do refer to fields as if they were variables, and compile them? Or do we supply a utility function to extract/insert data from a field? This needs more thought.

10. Screen Design

We did not have time to complete this chapter. This is one of the most difficult areas to design, since it is very visible and will no doubt be very important The Tutorial we have put together illustrates to lines along which we are working.

11. Helps and Tutorials

User-friendly systems have explanatory text stored on-line so the user does not need to memorize his manual or lug it around everywhere. The user may need help using a menu, understanding a screen, entering data in the correct format, or finding the correct arguments to use for commands. All this information should be available where and when the user needs it, at the press of a button (or a few buttons).

We would like to have so-called "active" helps, where the fields (or part of the field) for which help was requested exists in the help screen, so that the user can act on the help while reading it. For Menus and Screens, help should be displayed as "overlays" where possible, so that a portion of the screen is visible while help is displayed. This may seem trivial but it does help get rid of that awful discontinuity and the "lost" feeling that goes with it, when you ask for help and the screen is completely replaced by something else.

The layout of help-screens should be site-definable (within constraints).

We would like to hook help-screens together in a structure parallel to the application. The first help screen presented to the user may not answer his question. An intelligent help system should suggest further topics, and let the user select what he would like to see. The complete structure or connected helps can be used as a Tutorial for the application.

Much of the structure of the HELP system can be taken from the structure of the application, but it may be necesary for the programmer to go in and edit the "links" in the structure to define extra help-screens, or create relationships between topics not directly connected in the structure of the application as defined by the menu-tree.

APLPLUS/PC has a help system with some of these features, ISPF has tools for creating Tutorials. APLMON has an advanced "intelligent" help system. We have not had time to look closely at this problem yet, so we do not wish to say more about it, until we have had a chance to talk to people who have spent time investigating these problems.

12. Graphics

We have no experience of Applications which integrate textual screens with graphics, but IBM's APE product has this facility, and Bikuben want it (and it does sound nice)! Using GDDM (AP126), it is posible to handle screens in the same way as with AP124 (in fact AP126 is more advanced, but costs significantly more in terms of CPU consuption). With GDDM, a "text" screen can contain a single graphics field.

In theory, it should be possible to insert any SAGA (i.e. SUPERPLOT too) graph into such a field. We don't know enough about GDDM to know exactly how to do this.

Another popular method of generating GDDM graphics (besides SUPERPLOT) is called ICU - Interactive Chart Utility. It is an IBM product, integrated with GDDM. It should be possible to extract ICU-generated charts and put them on SHAPE screens as well.

IBM APE also has something called Image Design. This allows you to enter a multi-colour dot-matrix by filling in a field with 'Y' for a Yellow Dot, 'R' for a Red one, and so on. This is a simple way to enter Company Logos or other simple images. We should probably include an Image Design module in SHAPE, based on SAGA or possibly "raw" GDDM.

Thus, we want to allow users to define graphics screens into which they can stuff the SAGA, SUPERPLOT, ICU or ID image of their choice. SHAPE will automaticaly drive screens containing Graphics Fields using AP126, sticking to AP124 for other screens in the name of efficiency (though rumour has it that GDDM is getting better all the time). Lets hope that we are able to build a bridge to GDDM version 2, when it arrives.

13. Running the Application

SHAPE generates two types of end environments. If the application has no "variants", and is small enough, the user can specify a workspace name (one for each environment, if there is more than one). SHAPE generates a workspace, which will autostart the application.

For more complicated systems we recommend (and for multi-variant systems we demand) that the user specify that SHAPE generates "paging files". The function SHAPE-RUN (perhaps) in a public workspace starts the application. The function takes the name of the application as an argument, and it is possible to specify an optional phase (e.g. TEST) and a start-up sequence to be inserted into the command buffer.

As mentioned in chapter 7 (Menus), we will supply a workspace which is a "system selection menu", if the site wishes to have central control of access to applications.

13.1. Debugging

Shape, for all its advantages (ahem), effectively ties the hands of the programmer. The days of

```
(Qpack Qpnames Qread 1,23) Qreplace 2,23
or
)save temp26
```

are definitely gone. Much of the code written by the user resides in functions which the user doesn't even know the names of, and the functions he does know the names of may be spread across arbitrary components of a LOGOS paging file. We're going to have to sweat to get APL'ers to forgive us that!

There are a few things we can do to help:

- Allow the programmer to see what his menus and screens look like and test them as much as possible, during the design phase.
- Make it posible to test Transformation and Validation routines within SHAPE, without entering the application proper.
- Supply routines for editting SHAPE objects such as Transformations on the spot (in the crashed workspace), and putting the modifications back into SHAPE.
- For ordinary APL objects, supply a "snap" routine to detect changed ornew objects automaticaly, and put them into SHAPE (possibly asking the user where to put new objects).

Thus, if the (test) application crashes, we call up an editor immediately, highlighting the point in the code where the error occurred, allowing the user to inspect variables etc within the editor, fix the bug, and have the new code put back in its proper place.

For errors in Commands, Validations or other "compiled" functions, we must find the point where the error occurred in the SOURCE of the object, let the user edit the source, and re-compile the object on the spot. It should be possible to skip in and out of SHAPE from a crashed workspace, and to start an application up from within SHAPE.

13.2. Dummy Files and Conditional File Updates

Dummy files are files to which you can write with no risk of destroying anything. When you read from them you get random data, as specified in the File Design screen.

Conditional File updates are file writes where the system pauses before doing the write, and enters a fullscreen editor, displaying the file name, the record identifier (key), and the data about to be written. At this point the user can verify that the data is correct, and even edit it if there are trivial errors. This can be extremely useful when testing changes to file update routines! A file can be configured for Conditional File updates in the File Design screen.

13.3. Error Logging, Restartability

We'd also like to help the applications programmer when things go wrong. The programmer can enable "error logging" for his system. Error logging can either be a snap of the entire failing workspace to a holding library, or a record of the error message, the stack, other environmental parameters and possible copies of variables referenced on the failing line of code. If MAILBOX is installed at a site, a message can be sent to the author(s) of the systems, otherwise a message is saved in LOGOS, and the SHAPE PROFILE will alert the programmer the next time he enters SHAPE. For production systems, the System Administrator can be notified as well.

SHAPE applications are fully restartable (please don't mention sequential data sets, or DB2 dialogs). However, a system can be configured as non-restartable, in=which case we set a trap of '2001 D CLEAR', so no continue workspace is saved.

14. Generating Documentation

With all the structural information stored within SHAPE, we're going to be able to produce oodles of documentation - cross references between files, screens, data types - you name it! SHAPE can function as an "Active Data Dictionary" for APL - giving the Site Administrator an overview of all data used by APL applications, and helping programmers find the data they need.

Of course SHAPE should be able to provide function listings, display of data type definitions, screens and so on. It might make sense to interface to WSDOC, though it is not clear how WSDOC could use the additional information stored in the menu structure.

SHAPE should also have enough information to write most of the user- documentation by printing the menu structure, the screen layouts, and all the help- and tutorial screens.

We have not had time to design any particular report, but haven't been too worried about it since the big challenge seemed to be how to get the data into the system in an easy way - we will always be able to get it out again using APL.

15. Utility Functions

In our experience as APL consultants, we have seen our customers reinvent wheels (or even helped them to do sow, with out knowledge of good wheels seen at other sites). There are areas where we could save our customers a lot of time, and we would like to include these goodies within SHAPE. It might be that some of them (the print spooling system for example, should be developed as projects within the ZOO, rather than as part of the SHAPE project itself).

15.1. Print Spooling System

Every site has a requirement for creating reports and putting them into a queue (spooling system). Users can browse through the queue, and release one or more copies of selected reports, directing them to any printer attached to the system at that point, rather than doing so at the beginning of the function which creates the report. Reports can be left in the queue for other users to see as well. The system should remove the difference between HCPR and HSPR, and be easy to customize to add support for other print systems, such as Asynchronous printer-drivers, or home grown systems like IDIF (for 6670's) at ATP.

Included in such a system should be utilities for creating report files, browsing throgh them rather than printing them immediately, queue display and management, appending snapshots of screen images to a report (and more).

15.2. (Batch Task Control System)

This system is in many ways related to the Report Utilities, since Reports are often heavy jobs which get run in Batch Mode. We need tools for splitting off cpu-crunching routines as N-tasks, waiting for replies, or starting to browse a report being created by an N-task as soon as the N-task has finished formatting page 1.

We need tools for handling B-tasks: browsing through the request queue to check whether jobs have completed, looking at B-task crash workspaces, and helping build arguments to BTSUBMIT, in particular for setting up so-called BTASK Chains, which run a particular job at regular intervals, for example every third Tuesday.

15.3. Resource Consumption Monitor

Sites are getting more and more interested in being able to tell the Data Centre how much CPU each APL application is using. APL programmers need to know which parts of their applications are consuming most resources, so that they can decide where it might pay to optimize their code.

With QUAD-MF (release 19.1?) it should be easy to implement tools which let us report CPU consumption per application, menu, screen, command or user. We can prepare reports within APL or export them to RMF, as the site desires.

16. System Administrator Duties

The system administrator (who may in fact be several individuals) can have several areas of responsability, depending on how the site wants to make use of SHAPE:

1. Site Defaults, Installation Parameters, and so on:
   - The layout of Menu, Help, and "Empty" screens.
   - Default PF-keys and Commands
   - Site Profile (how to highlight errors, display box-drawing characters, etc).
   - Site MACROS
   - a Site Default EXIT routine, to be called on exit from all applications.
   - Site Data Types, and "Special Fields".
2. Enrolling programmers in SHAPE, and giving access to maintain applications
3. Putting systems into production, if a site wishes to use SHAPEs facilities for Strict Control with such systems. This work includes:
   - Checking that documentation is complete.
   - Checking that the application does not break any rules (these may be site-definable).
   - Backing off existing environments and putting the new ones into production.
   - Swearing at programmers when their applications fail, despite the System Administrator having set the "System No Crash" bit.
4. Controlling access to applications, either by changing the ACF interface routines so that they talk to an outside Access Control Facility such as RACF, or by maintaining the default access control system shipped with SHAPE.

5. Monitoring the Data Dictionary to check that programmers are sharing data sensibly, and not maintaining duplicate data bases, etc. Helping programmers searching for data or utility functions.

17. Macros

Macros allow the storage of Command Sequences, to make it possible to fire off frequently-used sequences with a single command (or perhaps even a PF-key). Viewpoint has MACROS, and SHAPEs MACROS will be indistinguishable from these. However, we would like to add a few Commands to give MACROS a little more power:

- Commands to set any line of any field on a screen to have a particular value:

  `SET CC TO 'LHG'` *or* `SET TEXT LINE 12 TO 'CHEERS / MORTEN'`

  This makes it possible to do (almost) everything in a MACRO, if you want to.

- An "INPUT" command, which displays the active screen, but locks all fields except the ones names in the argument to the INPUT command. Thus, a MACRO can pause at a given point, and request input for a particular field or fields.

- A "learn mode", where SHAPE records the actions you take, and creates a MACRO so you can replay it at a later date.

The ability to do "everything" in a MACRO, without ever having to display a screen, means that a MACRO can be fired off as a BATCH task. SHAPE applications can run in a BATCH mode, where all input must come from the Command Buffer. The task crashes if any other input is required. A Command called "BATCH" allows any Command Expression to be split off and run as a Batch task, either immediately or at a later time.

If MACROS tasks require features such as Screen Hardcopy, they cannot run as BATCH tasks, unless we do something devious based on NWS or similar techniques (someone better want it very badly).

18. Ideas

There are a number of topics which have not made it into a chapter of their own, because we have not had time enough to think about them. You could say the same for a lot of things which did make chapters of their own - these are topics which we thought we would have time to think about, when we started!

18.1. Integration with Viewpoint

Hopefully it is obvious by now that SHAPE could be of use to Viewpoint in two ways:

1. As a way to develop new Viewpoint modules.
2. As a tool to be made available to the End Users of Viewpoint, as a simple "Screen Painter", or as a way to implement more complicated "Auxiliary Commands".

It shouldn't be too hard to interface modules written in SHAPE to Viewpoint - the Viewpoint Screen driver seems to have quite a clean interface between modules, and so if SHAPE can have a Viewpoint interface which converts global Viewpoint variables of interest to SHAPE format on enetering a SHAPE module, and back again on exit, we may not need to do more to get the systems to work together. How was MB/3270 interfaced to start with?

SHAPE will want to make use of several parts of Viewpoint from the start, whether or not we have completed integration of the two systems:

- Profiles and Macros. We'd like to interface to (or simply steal) the code for maintaining Profiles an Macros.
- "External Interfaces". Rather than reinvent these, we'd like to use the new Viewpoint File Utilities.
- Reports - it might be nice to somehow call up Reports stored in Viewpoint from a SHAPE application. We have not included a report generator in our plans for SHAPE - if anyone wants this, they can buy VIEWPOINT.

18.2. Conversion Tools

We need to give users of existing fullscreen applications a way to move their screens into SHAPE with a minimum of effort. For example, we need a utility which, handed the name of a variable shared with AP124, is able to extract as much information as possible about the screen, and create a screen with those attributes in SHAPE.

18.3. Importing APE(2) Applications

It would be nice if we could import systems maintained in IBMs APE. The information stored in APE is a subset of that stored within SHAPE, so it should be possible to do quite a complete job. Any volunteers to write an APL2 to SAPL cross-compiler?

18.4. Producing BMS Macros

IBMs APE is able to produce screen defintions for assembler programmes using something called BMS Macros. We haven't a clue what these are, but ATP does it with a converted APE design tool, and there may be others who want this capability too.

18.5. Text Editor

SOmething a little more advanced than the DEL editor would be nice for editing the documentation, the help screens, etc.

18.6. PC Application Development

With growing use of pure PC or "hybrid" applications, we should investigate how SHAPE could be adapted to be a tool for people designing such systems. Hopefully we can get some ideas about this from the Global Limits crowd, or the other teams working on similar systems in Toronto.

It should be possible to extend SHAPE in such a way that some of the paths in the tree are 'on the PC'. During system generation, some of the compiled objects end up in a PC - and it should be up to the application developer to decide how much of his system he wants to run in each environment. Certainly, SHAPE should contain communications utilities which allow the Screen Definitions, and some of the Validation and Transformation code to reside on the PC.

The screen driver for the PC should be enhanced to take advantage of the facilities available there - mice, and other critturs that run around. We don't have enough knowledge to ignite much imagination in this area at the moment.

19. Desired Extensions to LOGOS

At this point, it seems clear that we will need three changes to LOGOS:

1. The Editor: BIKUBEN are expecting an ISPF-style editor (looks a lot like the IPSAAPE editor). We've been told that the new LOGOS editor will be something like this - if so we'd appreciate it if the completion of this editor could be given high priority. We would like to be able to call the editor to handle a Field Group on a screen, while SHAPE handles the rest, so that the editor can handle the editing of the code part of a Command or Transform, while SHAPE handles the other bits and pieces of the screen.
2. The introduction of a SHAPE Quota. Since each site will have different limits for LOGOS and SHAPE-only users, we need two quotas.
3. Some way of making it impossible for users with SHAPE-only access to use parts of LOGOS which they should not. For example, the may not edit other objects than SHAPE allows them to, or execute LOGOS commands from the editor. Rather than devising something special for SHAPE, it might make sense to introduce the concept of a "contstrained" LOGOS-user, who may only stay inside LOGOS while executing a script, is excluded from issuing commands, etc, and gets thrown out as soon as the script called by his LOGOs profile terminates.

Other requirements will no doubt surface as we nail the specs down a bit more.